Lecture 7: (05 July 2013)
Randomized Algorithms & Probabilistic Analysis

N. H. N. D. DE SILVA
DEPT. OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF MORATUWA
Announcement

- Mid-semester Quiz after the break!
- You can panic now (again)
Today’s Outline

- Randomized Algorithms
  - Introduction: Hiring Problem
    - Probabilistic analysis
    - Randomized algorithms
  - Randomized Quick Sort
  - Randomized Selection
  - Random Number Generation
References

▲ Mainly

▲ CLRS, Chapter 5 (pp. 114-128) and others

▲ Many online resources

▲ For detailed, in-depth coverage, read

▲ “Randomized Algorithms” by Motwani and Raghavan
Intro: Hiring Problem

Suppose you need to hire a new assistant

"It's a difficult position to fill. Someone who's smarter than me – and smart enough to pretend not to know it."
Intro: Hiring Problem

- Suppose you need to hire a new assistant
  - You get a candidate each day from an agency
  - You interview and decide to hire or not
Intro: Hiring Problem

- Suppose you need to hire a new assistant
- Need to pay the agency a fee for an interview
Intro: Hiring Problem

- Suppose you need to hire a new assistant
  - Hiring is more costly
    - Fire the current assistant
    - Pay a large hiring fee to the agency
You always want to have the best person
- If the interviewed person is better than the current assistant, then hire the new person
- You are willing to pay the resulting price
You want to estimate the price of strategy
Assume
- Candidates numbered $1,\ldots,n$
- After interviewing candidate $i$, determine if $i$ is the best seen so far
- Costs: for interviewing $c_i$ and for hiring $c_h$
Intro: Hiring Problem

...contd

HIRE-ASSISTANT (n)

best ← 0 // least qualified, dummy
for i ← 1 to n
  interview candidate i
  if candidate i is better than best
    best ← i
    hire candidate i

June 2013
N. H. N. D. de Silva
10
Intro: Hiring Problem  ...contd

- If \( m \) people hired, total cost \( O(n \ c_i + m \ c_h) \)
  - If \( c_i \) is small, can focus on \( (m \ c_h) \)

- Worst-case?
  - *Hire each person interviewed* (they come in increasing order of quality); total cost = \( n \ c_h \)
  - But reasonable to expect this will not happen
    - Yet we don’t know the order and we don’t have a control over that
    - What do we *expect to happen in an average case*?
Probabilistic Analysis

- Probabilistic analysis means
  - Analyzing problems using probability

- In such analysis
  - Use knowledge of or make assumptions about distribution of inputs
    - Without this, cannot use probabilistic analysis
  - Compute an expected cost or running time
    - Expectation taken over all possible inputs
      - averaging the cost/running time over that space
Probabilistic Analysis...contd

Example: Hiring problem/algorithm

Assume candidates come in random order

- Can compare any two and decide who is better
- There is a total order on the candidates
- Can rank each candidate with a unique number between 1 and $n$; $\text{rank}(i)$ denotes rank of $i$
- Convention: higher rank means better qualified
- The ordered list $<\text{rank}(1), \text{rank}(2), \ldots, \text{rank}(i)>$ is a permutation of the list $<1, 2, \ldots, n>$
- List of ranks equally likely to be any one of $n!$ permutations of numbers 1 through $n$
Randomized Algorithms

- As we saw, probabilistic analysis requires we know about the distribution on inputs
  - But this may not be so in many cases
- In our current HIRE-ASSISTANT algorithm
  - Candidates may seem to come randomly
    - But cannot know if this is correct or not
  - To have a randomized algorithm for this
    - Need to have greater control on interviewing order
    - What can we do?
Randomized Algorithms

...contd

- Randomized algorithm for hiring problem
  - Agency has $n$ candidates; list sent in advance
  - Each day we randomly choose whom to interview
  - A significant change!
    - Instead of assuming, we enforce random order

- Randomized algorithm
  - Behavior is determined by input and by values produced by a random-number generator
Randomized Hiring Algorithm

RANDOMIZED-HIRE-ASSISTANT (n)

randomly permute the candidate list

best $\leftarrow$ 0 // least qualified, dummy

for $i \leftarrow 1$ to $n$

interview candidate $i$

if candidate $i$ is better than best

best $\leftarrow i$

hire candidate $i$
Randomized Algorithms

- A randomized algorithm
  - An algorithm that makes random choices during execution
  - Random numbers used for making decisions
  - Behavior determined by a random-number generator (in addition to the input)
Basics

- Random decision making is introduced to reduce the chance of a worst-case scenario.

- Randomized algorithms have no worst-case behavior due to a particular input.
  - i.e., no bad inputs
  - But only bad random numbers!!
We compute *expected running-time* (the average case), considering probabilities when necessary.

Randomized strategy useful when:
- There are many ways we can proceed
- But, difficult to determine a way guaranteed to be good
If many alternatives are good, we can simply choose one randomly.

If we have to make many choices, a random selection of good and bad choices can be a good strategy if,…

Benefits of good choices outweigh the costs of bad choices.
Another Example

- Suppose a teacher wants to give a quiz in the class on the day a homework is due to make sure students did their own work
- But giving a quiz for every homework will consume time from limited class-time
- Practical solution might be to do this for 50% of homework
- How to decide when to give quizzes?
Another Example …contd

- Announcing in advance is not effective

- Giving un-announced quizzes on alternate homeworks → students will figure out

- Giving quizzes on “important” topics?
Another Example …contd

- Easiest is to *flip a coin to decide*
Easiest is to *flip a coin to decide*

- 50% probability for a quiz on a homework
- Expected number of quizzes = 1/2 of the number of homeworks
- It is possible, but unlikely, that there is no quiz for the whole semester (or the opposite) – unless the coin is biased
- *a randomized algorithm*
Randomized strategy particularly useful when faced with a malicious “adversary”

- Will deliberately try to feed a bad input to the algorithm
- Randomness commonly used/applied in cryptography
- Issue: pseudo-random number generator
Two types of randomized algorithms

1. Always gives the correct answer; may require some time/resources to execute → *Las Vegas algorithms*

2. Can complete quickly (bounded resource usage) but the answer may not be 100% accurate → *Monte Carlo algorithms*

Special complexity classes, analysis
Randomized Quick Sort

- Recall Quick Sort from Lecture 2
- (See also pp. 170-185, Ch. 7 of CLRS)

Randomized Quick Sort Version 1

- Before sorting the array, randomly permute the elements
  - Enforces the property that every permutation is equally likely
  - Makes the running time independent of the original input ordering
Randomized Quick Sort Version 2

Modifying the original PARTITION procedure, perform a randomized-PARTITION

Slight changes to original procedures

At each step, before partitioning, exchange $A[p]$ with another element chosen randomly from $A[p...r]$
Random Numbers?

- Assume: we have a random-number generator of the form \( \text{RANDOM}(a,b) \)
  - Returns a random integer between \( a \) and \( b \), inclusive, with each being equally likely.

- In practice, true randomness cannot be achieved with a computer.

- Most programming environments provide pseudo-random number generators.
Randomized-QuickSort, V2

Input: Unsorted sub-array $A[p..r]$
Output: Sorted sub-array $A[p..r]$

**RAND-QUICKSORT** $(A, p, r)$

if $p < r$

then $q \leftarrow$ **RAND-PARTITION**$(A, p, r)$

**RAND-QUICKSORT** $(A, p, q-1)$

**RAND-QUICKSORT** $(A, q+1, r)$
Randomized-Partition Algorithm

Input: same as for PARTITION( )
Output: same as for PARTITION( )

**RAND-PARTITION** \((A, p, r)\)

\[ i \leftarrow \text{RANDOM}(p, r) \]
Exchange \(A[p] \leftrightarrow A[i]\)
return **PARTITION**\((A, p, r)\)
Complexity Analysis

- Worst-case: discussed earlier
  - Running time $\Theta(n^2)$

- Average-case (expected) running time of randomized Quick Sort is $\Theta(n \lg n)$
  - Details: pp. 180-184 in CLRS
Average-case Analysis

- Intuitively, average-case running time of randomized Quick Sort is $\Theta(n \lg n)$
  - partitioning splits the array such that a fraction of elements are on one side
  - recursion tree has depth $\Theta(\lg n)$
  - $\Theta(n)$ work is performed at these $\Theta(\lg n)$ levels
Average-case Analysis

...contd

Observations for precise analysis

- Value $q$ returned by PARTITION depends only on the rank of $x = A[p]$ among $A[p...r]$.
- The rank of $x$, $\text{rank}(x)$, in a set is the number of elements less than or equal to $x$ in the set.
- Due to swapping with a random element first, $\text{rank}(x) = i$ for $i=1,2,...,n$ with probability $1/n$.
- Assumptions: $n = r - p + 1$ (# elements in $A[p...r]$), elements are distinct.
Average-case Analysis

...contd

Observations for precise analysis ... contd

- If $\text{rank}(x)=1$: $q=j=p$ is returned, low side of partition contains 1 element $A[p]$
  - This event occurs with probability $1/n$
- If $\text{rank}(x)=2$: the smallest element will go to $A[p]$; $q$, low side, probability same as above
- If $\text{rank}(x) > 2$: low side of partition has $i$ elements; probability $=1/n$ for each $i=2,\ldots,n-1$
Average-case Analysis

...contd

- Size of low side of partition \((q-p+1)\) is
  - 1 with probability \(2/n\)
  - \(i\) with probability \(1/n\) for \(i=2,3,\ldots,n-1\)

- Recurrence for the average (expected) running time of randomized Quick Sort

\[
T(n) = \frac{1}{n} \left( T(1) + T(n-1) + \sum_{q=1}^{n-1} (T(q) + T(n-q)) \right) + \Theta(n)
\]
Average-case Analysis

...contd

- Can simplify and re-write this as

\[ T(n) = 2 \sum_{k=1}^{n-1} T(k) + \Theta(n) \]

- Solve the recurrence (substitution method)

\[ T(n) \leq a \cdot n \cdot \log n + b \]

- Average running time is \( O(n \log n) \)
Randomized Selection

- Discussion based on CLRS pp. 213-222
  - In Chapter 9, Medians and Order Statistics

- Selection problem
  - Input: a set $A$ of $n$ distinct numbers and a number $i$ such that $1 \leq i \leq n$
  - Output: the element $x$ of $A$ that is larger than exactly $i-1$ other elements of $A$
Randomized Selection

- Selection problem can be solved in $O(n \lg n)$ time
  - Sort the numbers first and then index the $i$-th element in the array

- But can be done faster, in $O(n)$ average-time, using a randomized algorithm
Randomized-Select Algorithm

**RAND-SELECT**\((A, p, r, i)\)

if \(p = r\)

then return \(A[p]\)

\(q \leftarrow \text{RAND-PARTITION}(A, p, r)\)

\(k \leftarrow q - p + 1\)

if \(i = k\) then return \(A[q]\) // pivot is the answer

elseif \(i < k\)

then return **RAND-SELECT** \((A, p, q-1, i)\)

ever return **RAND-SELECT** \((A, q+1, r, i - k)\)
Random Number Generation

- We consider the generation of *pseudorandom numbers*
  - Satisfy most statistical properties of random numbers and appear to be random
- In many cases, we need a *sequence of random numbers*
  - So use of the system clock may not work
  - Numbers should look independent
Random Number Generation

- Standard method
  - *Linear congruential generator*
  - First described by Lehmer in 1951

- Numbers $x_1, x_2, \ldots$ are generated satisfying
  $$x_{i+1} = a \times x_i \mod m$$

- $x_0$ is called the *seed* (must be given, $\neq 0$)

- $a$ and $m$ to be selected suitably
Random Number Generation

- Standard method
  - E.g., if \( m=11 \), \( a=7 \) and \( x_0=1 \), then we get 7, 5, 2, 3, 10, 4, 6, 9, 8, 1, 7, 5, 2, …
  - Here, after \( m-1=10 \), the sequence repeats
  - If \( m \) is a prime, then there are always choices for \( a \) that give a full period of \( m-1 \)
Random Number Generation

- Standard method
  - For some $a$, this will not happen
  - e.g., if $m=11$, $a=5$ and $x_0=1$, then we get a sequence with a shorter period
    
    $5, 3, 4, 9, 1, 5, 3, 4, \ldots$
  
  - Generally, the 31-bit prime $m=2^{31}-1 = 2,147,483,647$ and $a = 7^5 = 16,807$ are commonly used
    
    (this $a=7^5$ gives a full period generator)
Random Number Generation

- When $m$ is a prime, $x_i$ is never 0
- Sometimes we need a random real number in the between 0 and 1
  - This is obtained easily by dividing the above formula by $m$
  - Normalize to get 0 and 1
Conclusion

- Randomized algorithms
  - Introduction, Hiring problem
  - Randomized Quick Sort, Selection, Random Number Generation

- Next time: Graph algorithms
References

- Randomized Algorithms & Probabilistic Analysis [CLRS Chapter 5]

- The lecture slides are based on the slides prepared by Prof. Sanath Jayasena for this class in previous years.